Can a Decentralized Metadata Service Layer Benefit Parallel Filesystems?

Vilobh Meshram, Xavier Besseron, Xiangyong Ouyang
Raghunath Rajachandrasekar, Ravi Prakash Darbha
Dhabaleswar K. Panda

Network-Based Computing Laboratory
Department of Computer Science & Engineering
The Ohio State University

Presentation Outline

- Introduction & Motivation
- Problem Statement
- Design & Implementation of DUFS
- Performance Evaluation
- Conclusions & Future Work

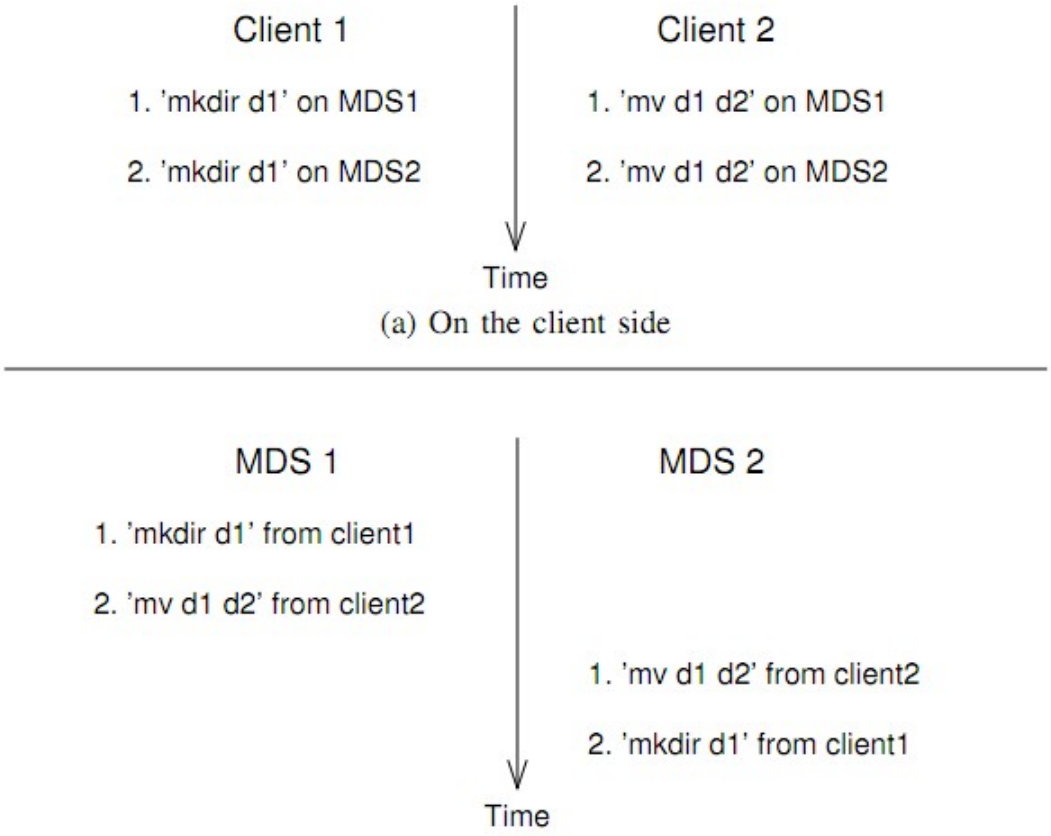
Why is Metadata Important?

- Metadata accessed when files are opened, closed, searched, deleted...
- Maintains a global directory hierarchy
- Over 75% of all filesystem calls require access to file metadata
- Metadata operations fall in the critical path of a broad spectrum of applications.
- Efficient management of metadata significantly improves throughput

Motivation

- Bandwidth usually improved by aggregation, striping, resource sharing, etc.
- Metadata Server(MDS) Bottlenecks
  - Single primary MDS
  - Contention increases as #clients grow
  - Fail-over MDS becomes operational when primary fails
  - Need to have a decentralized solution!
- Managing multiple MDSs
  - Maintaining several copies of directory hierarchy gets tricky!
  - Atomic operations - need for a global lock => hurts latencies
  - Guarantee in-order metadata service
  - Consistency concerns

Managing Multiple MDSs

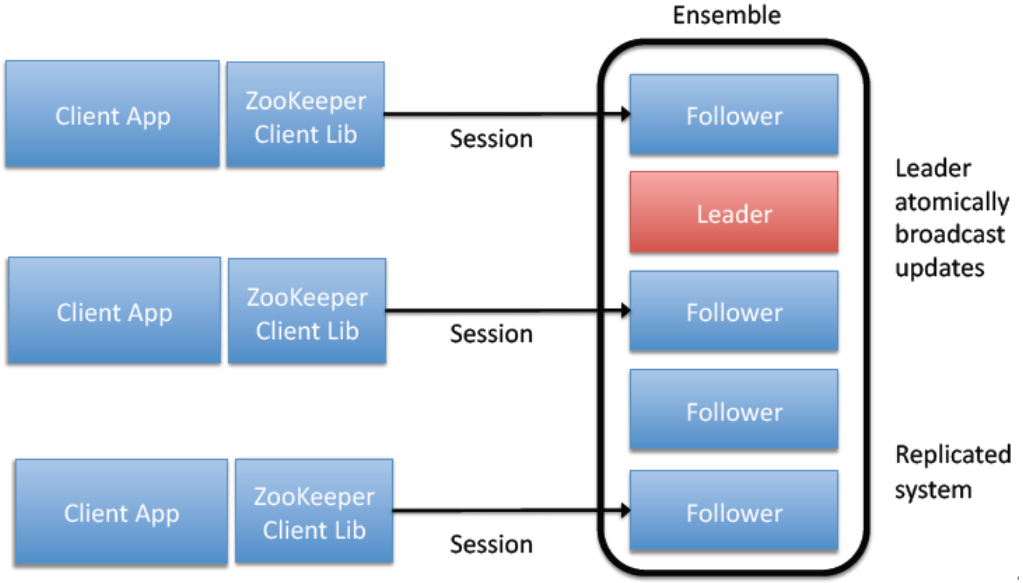


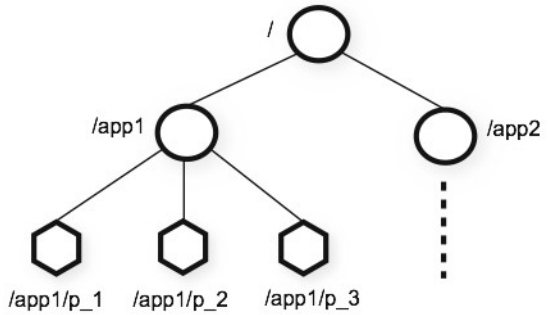
- Lack of coordination – results in an inconsistent state across MDSs

Distributed coordination schemes required to achieve consistency!

The ZooKeeper Service

- Open-source distributed coordination service
- Distributed processes coordinate through shared hierarchical namespace
- Namespace contains special nodes – Znodes
- Multiple servers – replicates the namespace
- Modifications to namespace – *atomic* and *strictly* ordered





Problem Statement

- Can a distributed coordination service (such as ZooKeeper) be incorporated into parallel filesystems to scale metadata processing throughput?
- What will be the *performance impact* of such a decentralized metadata service layer?
- Will this service layer maintain the *consistency* and *reliability* of the filesystem?

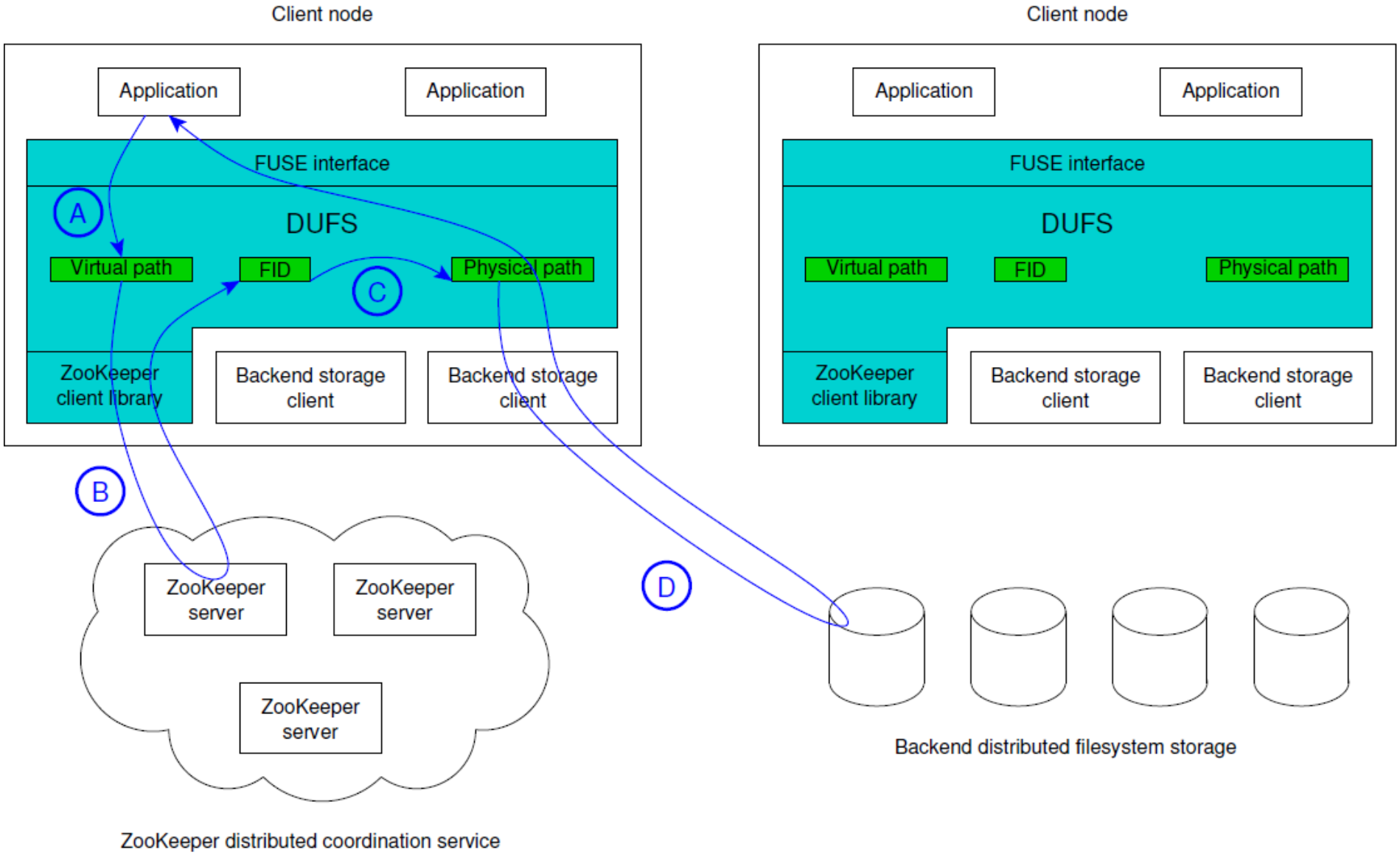
Design and Implementation

Distributed Union File System(DUFS)

- Design Principles
  - No single point of Metadata Service
  - Combine several mounts and provide a POSIX-compliant interface
  - Clients schedule metadata operations across multiple filesystems
  - Provide consistency and order guarantee

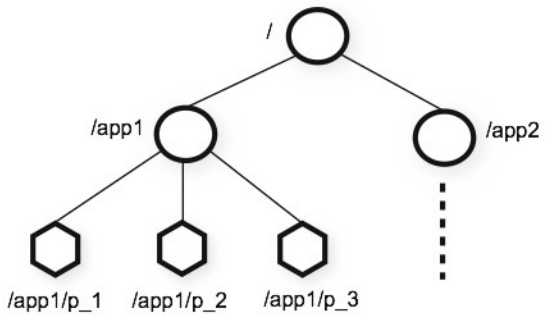
- Design Components
  - FUSE clients to provide a single POSIX interface abstraction
  - Zookeeper coordination service used to manage metadata
  - File Identifier (FID) allocation
  - Deterministic FID mapping function
  - Data management on multiple underlying mounts

DUFS Design

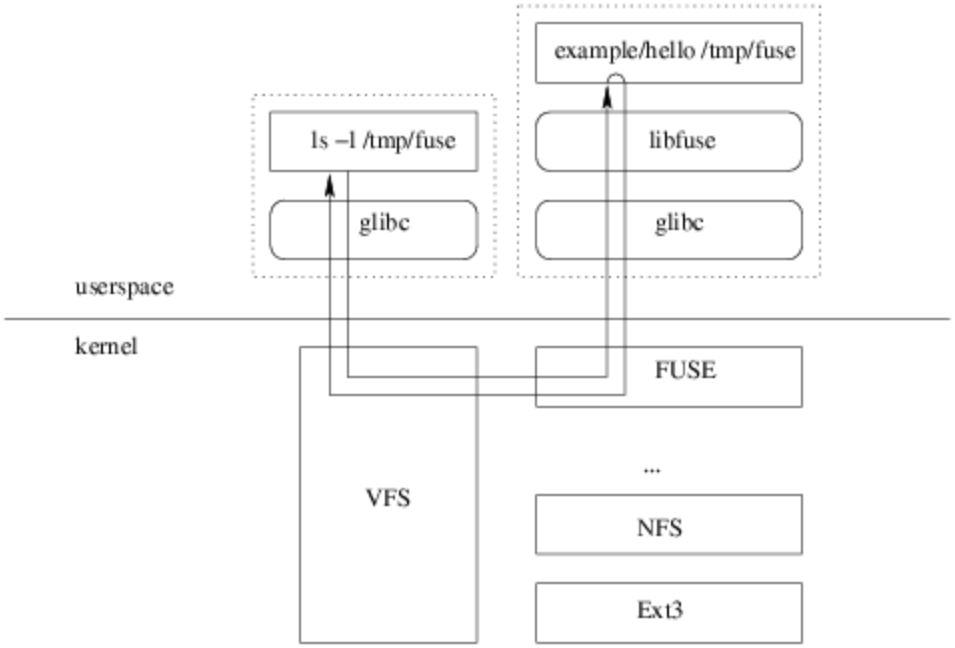

Client node
Client node
Application
Application
Application
Application
FUSE interface
FUSE interface
A
DUFS
DUFS
Virtual path
FID
Physical path
Virtual path
FID
Physical path
C
ZooKeeper client library
Backend storage client
Backend storage client
ZooKeeper client library
Backend storage client
Backend storage client
B
D
ZooKeeper server
ZooKeeper server
ZooKeeper server
Backend distributed filesystem storage
ZooKeeper distributed coordination service

Metadata Management with ZooKeeper

- Used to address consistency threats posed by distributed MDS
- Synchronous Zookeeper API used
- Virtual filesystem hierarchy replicated within Zookeeper
- A unique Znode created for each file / folder in DUFS
- Znode custom field used to store FID (if a file is being represented)
- All information kept in-memory – high operation throughput
- Downside – higher memory consumption



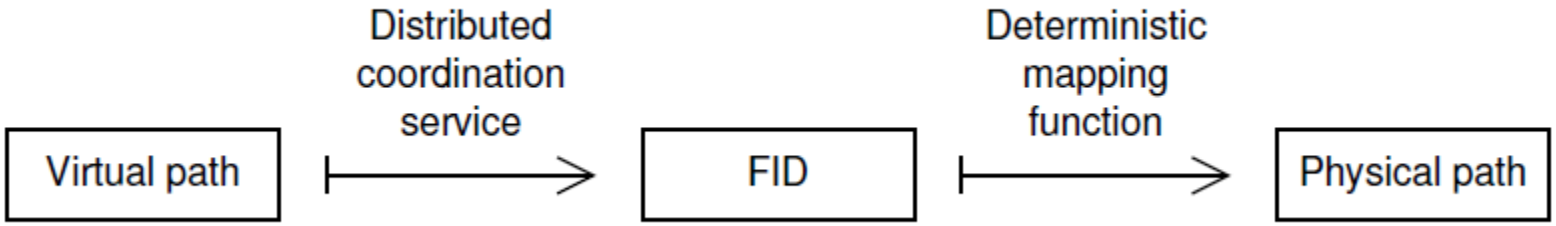
FUSE-based Filesytem Interface



- DUFS provides a POSIX interface, just as any classical FS
- Support for standard system calls
  - mkdir, create, open, symlink, rename, stat, readdir, rmdir, unlink, truncate, chmod, access, read, write
- DUFS exposes a virtual path to the client/application

File Identifier (FID)

- Unique for each newly created file
- 128-bit length : 64-bit client ID + 64-bit client-specific file counter



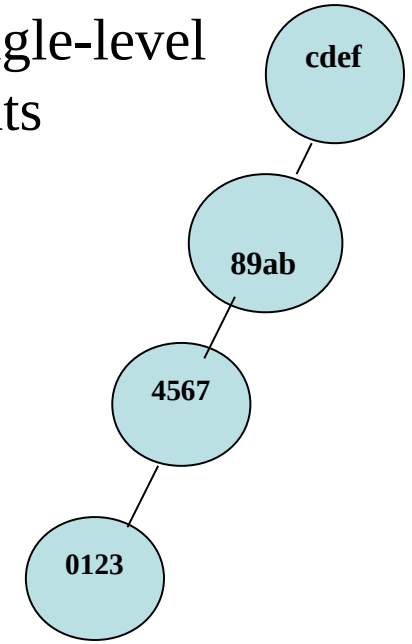
- Used to deduce the physical location of the file
- FID also used as filename in the underlying filesystem
- Modifications to contents of file does not disturb FID

Deterministic FID Mapping Function

Physical Data Storage

- Physical filename – Hex equivalent of FID
- Hex representation – 4 path components to avoid single-level congestion : 1filename + 3 path hierarchy components

FID:	0123456789abcdef
Physical filename:	cdef / 89ab / 4567 / 0123



- Directory hierarchy kept static across mount-points
- Clients need not communicate with any central component

stat() Algorithm

```
Get the virtual path of the file/directory
Get the corresponding Znode with ZooKeeper
if Znode does not exist then
    return 'No such file or directory' error code
else
    ZooKeeper returned the data field (type, FID, ...)
    if Znode type is directory then
        Fill struct stat with info stored in ZooKeeper
        return struct stat
    else
        Compute the physical location
        Compute the physical path
        Perform stat() on the physical file
        return struct stat
    end if
end if
```

Reliability Concerns

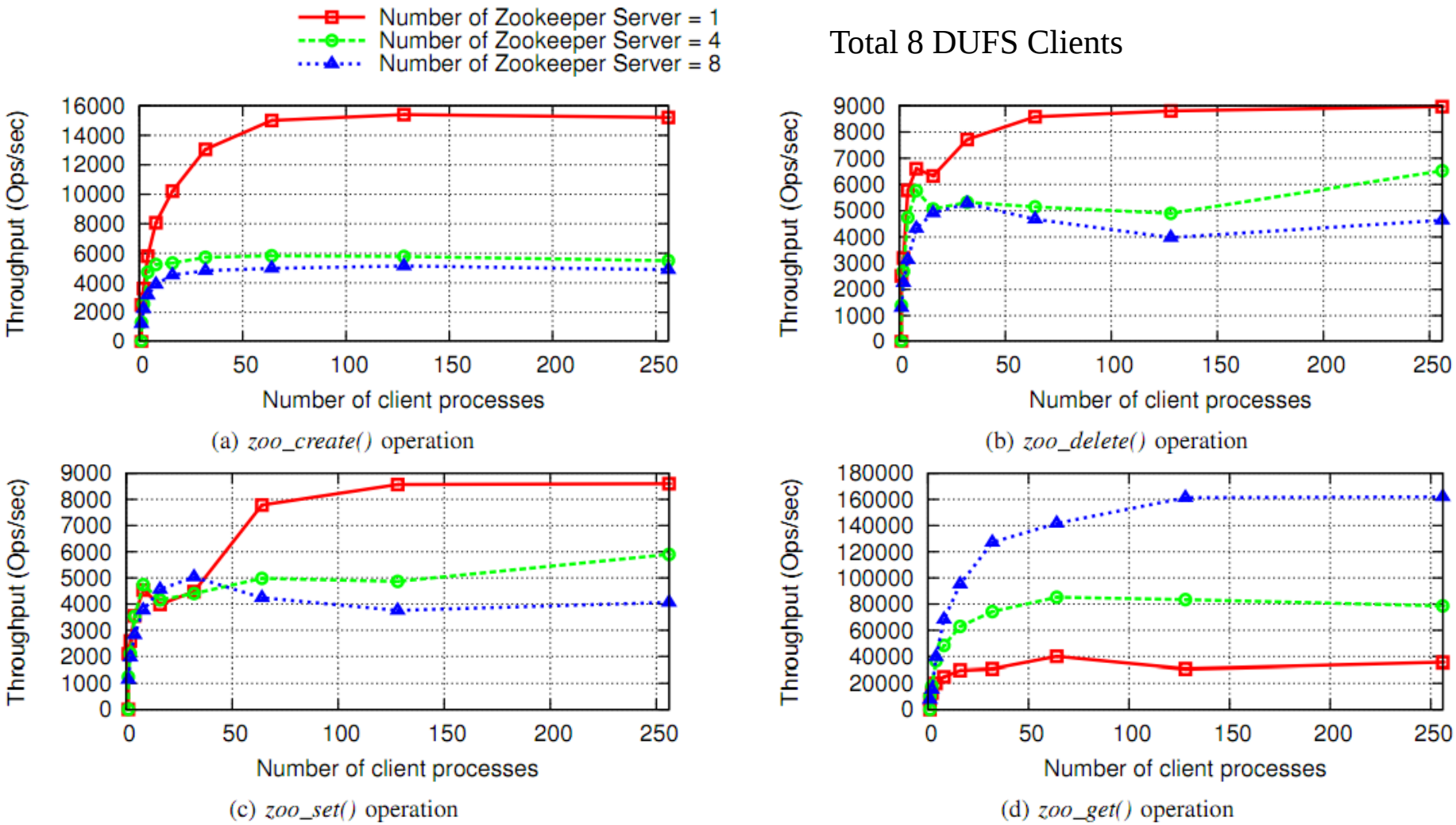
- DUFS is *stateless*!
- Metadata managed by Zookeeper
  - Information duplicated across servers
  - Requires a majority of the servers to stay alive
  - No threat due to in-memory storage – data checkpointed to disk
- Data managed by backend storage
  - Distributed filesystems such as Lustre provide fault-tolerance – failover servers, data duplication, etc.

Performance Evaluation

Experimental Environment

- 512-core Linux cluster with Intel Xeon CPUs
- 6GB memory / node
- Multiple Lustre instances
  - OSS v1.8.3
  - 12-disk RAID-0 configuration
- Multiple PVFS instances (v2.8.2)
- FUSE v2.8.5
- MDTEST benchmark suite
  - Directory tree fan-out factor 10
  - Directory hierarchy depth 5
  - 10 files per directory
- ZooKeeper v3.3.3(upto 8 servers)

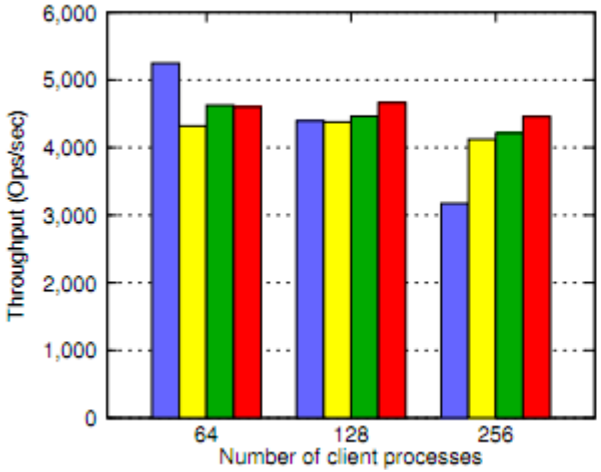
Zookeeper Scalability Analysis



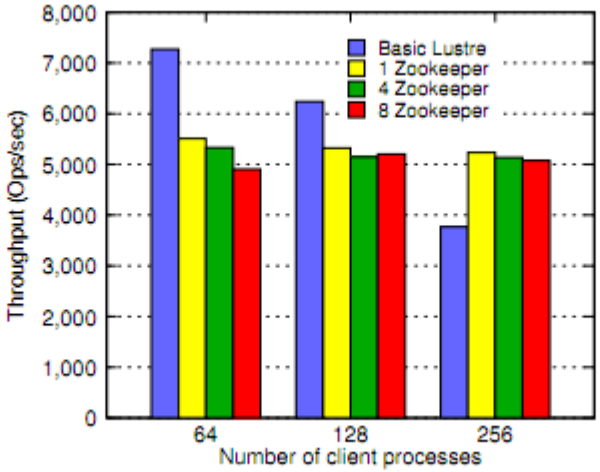
High-throughput for Read-Dominant workloads!

Varying ZooKeeper Servers – Directory ops

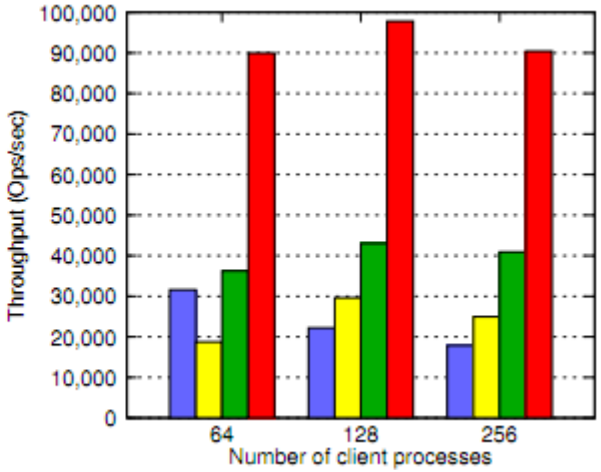
Up to 8 DUFS Clients



(a) Directory creation

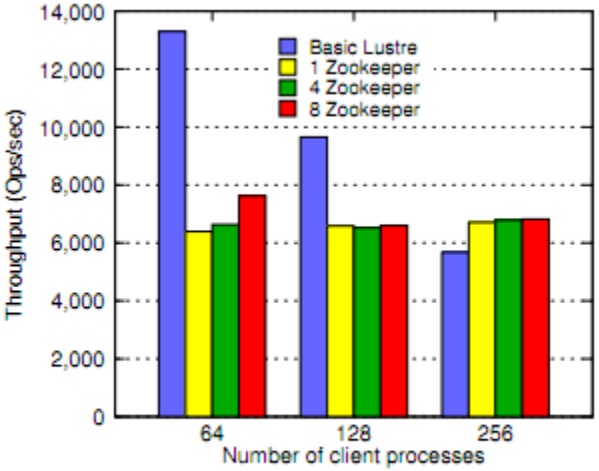


(b) Directory removal

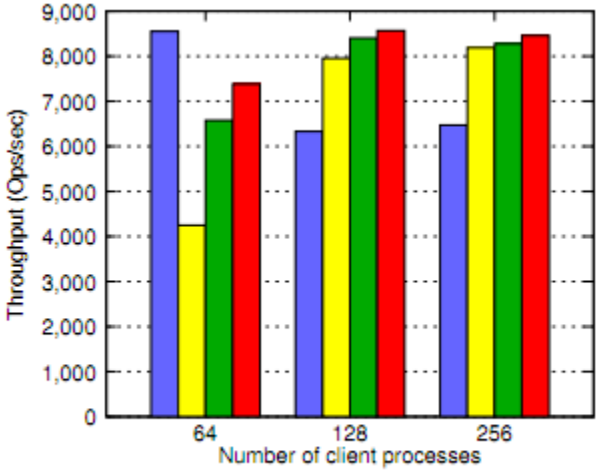


(c) Directory stat

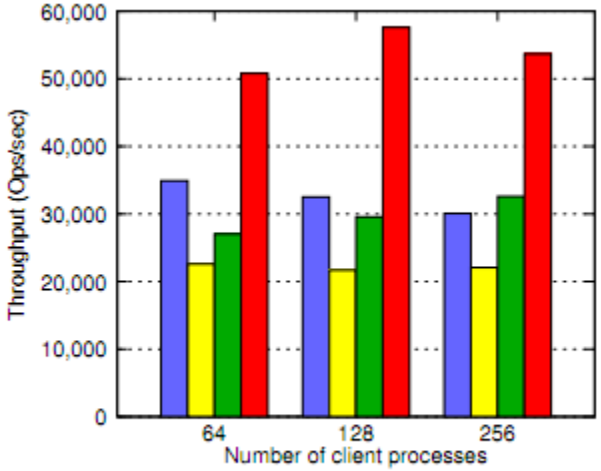
Varying ZooKeeper Servers – File ops



(d) File creation



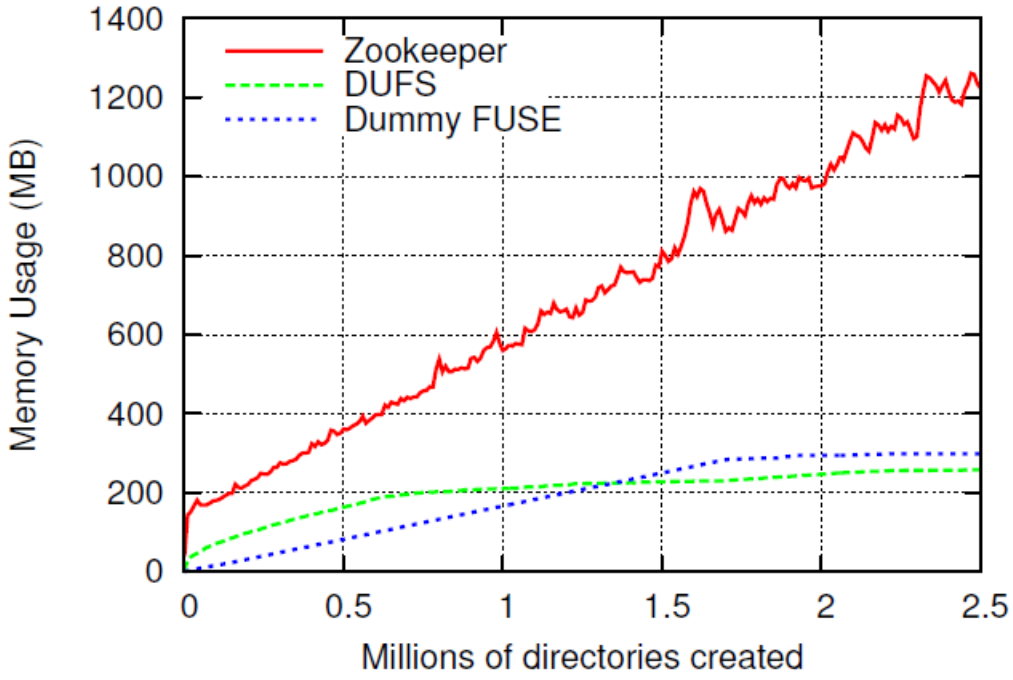
(e) File removal



(f) File stat

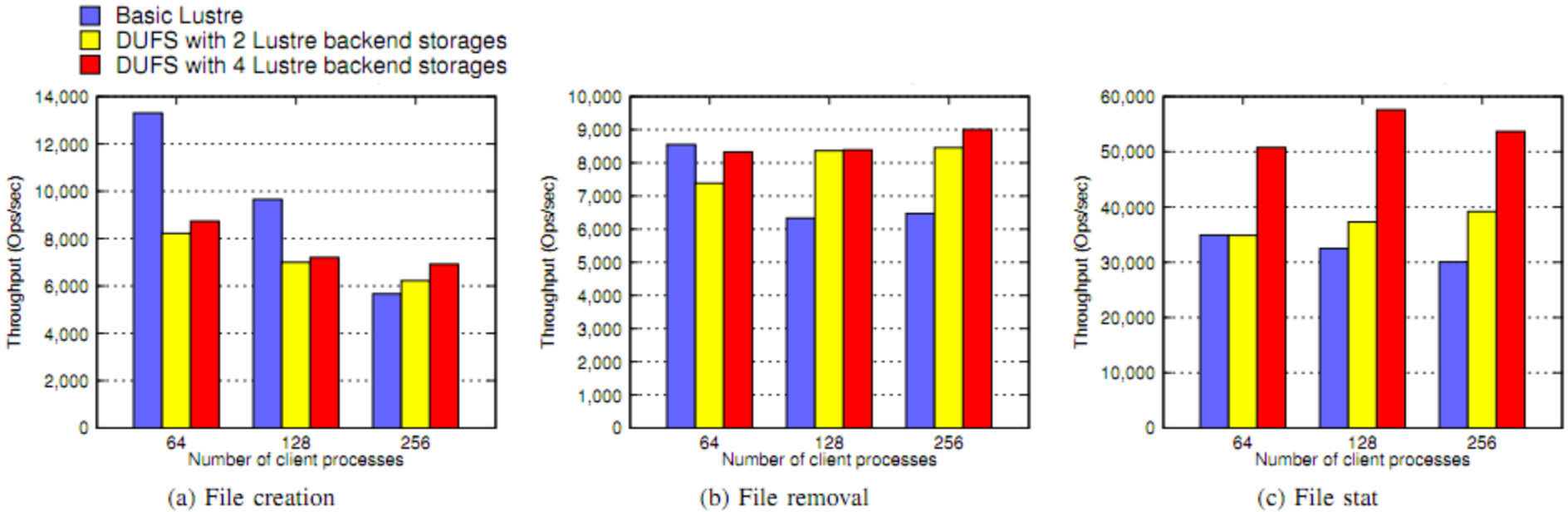
8 ZooKeeper servers ideal for read-cum-write workloads

Memory Usage Analysis



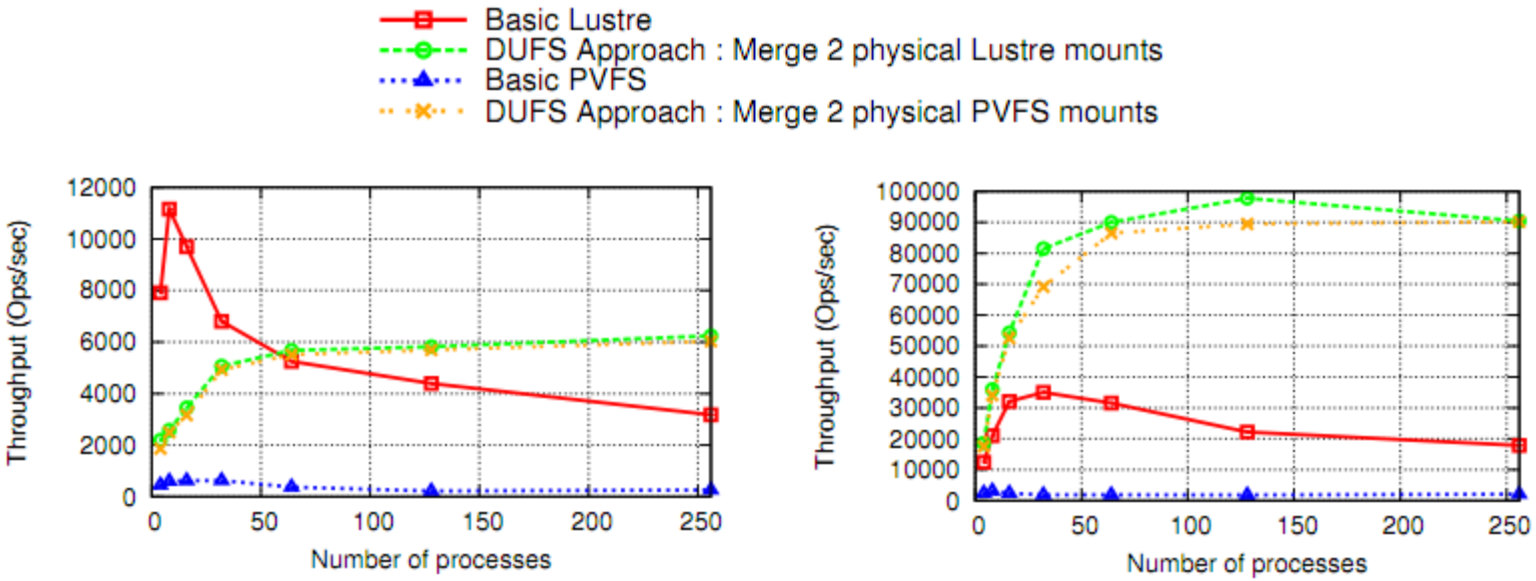
- All ZooKeeper data kept in-memory
- Memory usage proportional to #znodes (directories/files) created
- About 417MB memory required to store 1 million files/dirs
- ZooKeeper server with 24GB memory - ~60 million files/dirs

Varying Backend Mountpoints



File stat throughput improved by 37% with 256 parallel processes

Comparison with Lustre/PVFS2 – Dir ops

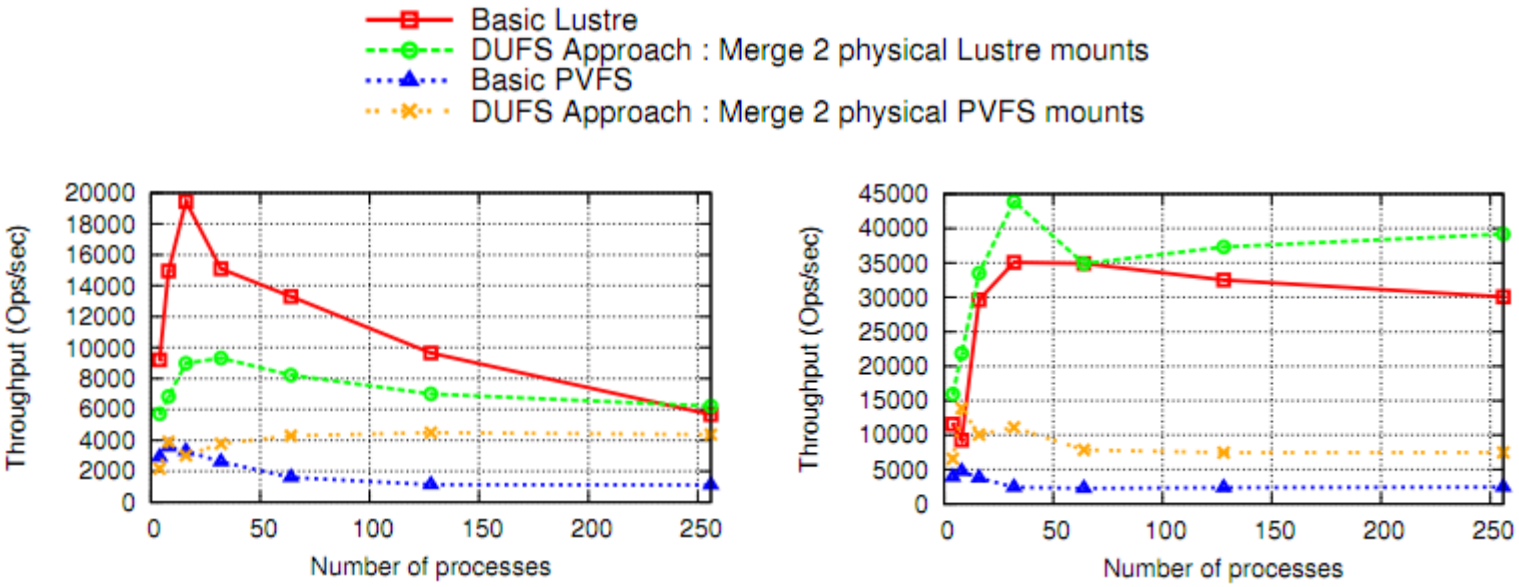


Directory Creation

Directory Stat

DUFS create outperforms Lustre / PVFS2 by 1.9x / 23 respectively at 256 procs

Comparison with Lustre/PVFS2 – File ops



File Creation

File Stat

DUFS stat outperforms Lustre / PVFS2 by 1.3x / 3x respectively at 256 procs

Conclusion & Future Work

- Scaling metadata performance is more complex than scaling raw I/O
- Designed a prototype filesystem to demonstrate the benefits
- Studied memory and throughput trends using the prototype
- We plan to study dynamic expansion of backend storage
- Study the trade-offs between dir hierarchy replication and striping

Distributed metadata service can benefit parallel filesystems without compromising consistency & reliability!

Thank you!

Network-Based Computing Laboratory

http://nowlab.cse.ohio-state.edu

{meshram, besseron, ouyangx, rajachan, darbha, panda}

@cse.ohio-state.edu